International Journal of Computer Science Engineering and Information Technology Research (IJCSEITR)
ISSN(P): 2249-6831; ISSN(E): 2249-7943
Vol. 5, Issue 1, Feb 2015, 55-62
© TJPRC Pvt. Ltd

MULTI-SECURE BINDER FOR DISMISSAL IN OS - LEVEL VIRTUAL MACHINES ENVIRONMENT

GIRISH WADHWA

Alamuri Ratnamala, Institute of Engineering and Technology, Thane, Maharashtra, India

ABSTRACT

The virtual machine provides the software implementation process where the task created and closed. Basically virtual machine has two types that are process virtual machine and system virtual machine. It provides the environment for single program execution and system platform. The problem of virtual machine is there is no malicious activity detection only the benign updates send to the Virtual Machine host environment. In this paper we propose the Virtual Machine Commitment system and also based on the enhanced and securing method than the existing process is called compressing process for the malicious clusters of the OS flow of the Virtual Machines. For detecting the malicious cluster the SECOM method will be used. It reduces the false-positive rate when identifying malicious clusters. OS-level virtual machines have the reduced hardware requirements and they are created during the need and they are closed after the usage or after the task completion. But there is an issue of detecting the malicious activity of the user. In the prior work, they developed the secure commitment method which provides the efficient malicious attack detection process but they not support the multi-VM environment. In this paper, we propose the multi-secure binder to commit the malicious activity. As additional we propose a concept of compressing the file after detecting the malicious user files. Hence it won't affect the system but it will be provide support the other supportive files in the system which is added as an additional file to that compressed file

KEYWORDS: Virtual Machine, Virtual Machine Host Environment, OS-Level

Objectives

- The VM commitment system commits the benign updates to host environment.
- It eliminates the malicious state changes in the VM host.
- It grouping the information flow into object for labeled the compromised object.

INTRODUCTION

A virtual machine (VM) is a software implementation of a machine (i.e. a computer) that executes programs like a physical machine. Virtual machines are separated into two major classifications, based on their use and degree of correspondence to any real machine: A system virtual machine provides a complete system platform which supports the execution of a complete operating system (OS). These usually emulate an existing architecture, and are built with the purpose of either providing a platform to run programs where the real hardware is not available for use (for example, executing software on otherwise obsolete platforms), or of having multiple instances of virtual machines leading to more efficient use of computing resources, both in terms of energy consumption and cost effectiveness (known as hardware virtualization, the key to a cloud computing environment), or both. A process virtual machine (also, language virtual machine) is designed to run a single program, which means that it supports a single process. Such virtual machines are usually closely suited to one or more programming languages and built with the purpose of providing program portability and flexibility (amongst other things). An essential characteristic of a virtual machine is that the software running inside is limited to the resources and abstractions provided by the virtual machine—it cannot break out of its virtual environment. A virtual machine was originally defined by Popek and Goldberg as "an efficient, isolated duplicate of a real machine". Current use includes virtual machines which have no direct correspondence to any real hardware Multiple VMs each running their own operating system (called guest operating system) are frequently used in server consolidation, where different services that used to run on individual machines to avoid interference are instead run in separate VMs on the same physical machine. The desire to run multiple operating systems was the original motivation for virtual machines, as it allowed time-sharing a single computer between several single-tasking Operation Systems. In some respects, a system virtual machine can be considered a generalization of the concept of virtual memory that historically preceded it. IBM's CP/CMS, the first systems to allow full virtualization, implemented time sharing by providing each user with a single-user operating system, the CMS. Unlike virtual memory, a system virtual machine allowed the user to use privileged instructions in his code. This approach had certain advantages, for instance it allowed users to add input/output devices not allowed by the standard system.

As technology evolves virtual memory, in respect to virtualization, will utilize the technologies of memory over commitment to manage the memory sharing between multiple virtual machines on one physical computer. On a related note, sometimes it is possible to share those memory pages that have identical contents among multiple virtual machines running on the same physical machine, mapping them to the same physical page, by a technique known as Kernel Same Page Merging. This is particularly useful for read-only pages, such as those ones containing code segments, especially in the case of multiple virtual machines running the same or similar software, such as the Operating System, software libraries, web server, and middleware components.

The guest OS’s do not have to be compliant with the hardware making it possible to run different OS’s on the same computer (e.g., Microsoft Windows and Linux, or older versions of an OS to support software that has not yet been ported to the latest version). The use of virtual machines to support different guest OS’s is becoming popular in embedded systems; a typical use is to support a real-time operating system at the same time as a high-level OS such as Linux or Windows. Another use is to sandbox an OS that is not trusted, possibly because it is a system under development. Virtual machines have other advantages for OS development, including better debugging access and faster reboots. A special case of process VMs are systems that abstract over the communication mechanisms of a (potentially heterogeneous) computer cluster. Such a VM does not consist of a single process, but one process per physical machine in the cluster. They are designed to ease the task of programming concurrent applications by letting the programmer focus on algorithms rather than the communication mechanisms provided by the interconnect and the OS. They do not hide the fact that communication takes place, and as such do not attempt to present the cluster as a single machine. Malicious software (or malware) is one of the most pressing and major security threats facing the Internet today. Anti-virus companies typically have to deal with tens of thousands of new malware samples every day. Because of the limitations of static analysis, dynamic analysis tools are typically used to analyze these samples, with the aim of understanding how they behave and how they launch attacks. This understanding is important to be able to develop effective malware countermeasures and mitigation techniques. Researchers have extensively studied the malware problem domain. One line of research has focused on the extent to which certain classes of malware have penetrated the Internet. For example, there have been studies that quantify the size of botnets, the number of executable infected with spyware, and the number of malicious web sites that launch drive-by downloads. Another line of research deals with tools to collect and study malware. Here, researchers have introduced various forms of honey pots, static analysis techniques, and dynamic monitoring tools. Finally, there are proposals to detect and remove malware once it has infected a machine, using either signature-based or behavior-based approaches. Proliferation of malware poses a major threat to modern information technology. According to a recent report by Microsoft, every third scan for malware results in a positive detection. Security of modern computer systems thus critically depends on the ability to keep anti-malware products up-to-date and abreast of current malware developments. This has proved to be a daunting task. Malware has evolved into a powerful instrument for illegal commercial activity, and a significant effort is made by its authors to thwart detection by anti-malware products. As a result, new malware variants are discovered at an alarmingly high rate, some malware families featuring tens of thousands of currently known variants. Malware is one of the most serious security threats on the Internet today. In fact, most Internet problems such as spam e-mails and denial of service attacks have malware as their underlying cause. That is, computers that are compromised with malware are often networked together to form botnets, and many attacks are launched using these malicious, attacker-controlled networks. With the increasing significance of malware in Internet attacks, much research has concentrated on developing techniques to collect, study, and mitigate malicious code. Without doubt, it is important to collect and study malware found on the Internet. However, it is even more important to develop mitigation and detection techniques

based on the insights gained from the analysis work. Unfortunately, current host-based detection approaches (i.e., anti-virus software) suffer from ineffectivedetection models. These models concentrate on the features of a specific malware instance, and are often easily evadable by obfuscation or polymorphism. Also, detectors that check for the presence of a sequence of system calls exhibited by a malware instance are often evadable by system call reordering. In order to address the shortcomings of in effective models, several dynamic detection approaches have been proposed that aim to identify the behavior exhibited by a malware family. Although promising, these approaches are unfortunately too slowto be used as real-time detectors on the end host, and they often require cumbersome virtual machine technology. Host-based malware detectors have the advantage that they can observe the complete set of actions that a malware program performs. It is even possible to identify malicious code before it is executed at all. Unfortunately, current host-based detection approaches have significant shortcomings. An important problem is that many techniques rely on *i*neffective models. Ineffective models are models that do not capture intrinsic properties of a malicious program and its actions but merely pick up artifacts of a specific malware instance. As a result, they can be easily evaded. For example, traditional anti-virus (AV)

Programs mostly rely on file hashes and byte (or instruction) signatures. Unfortunately, obfuscation techniques and code polymorphism make it straightforward to modify these features without changing the actual semantics (The behavior) of the program. Another example are models that capture the sequence of system calls that a specific malware program executes. When

These system calls are independent, it is easy to change their order or add irrelevant calls, thus invalidating the captured sequence. Malware detection approach is both effectiveand efficient, and thus, can be used to replace or complement traditional AV software at the end host. For this, we first generate effective models that cannot be easily evaded by simple obfuscation or polymorphic techniques. More precisely, we execute a malware program in a controlled environment and observe its interactions with the operating system. Based on these observations, we generate fine-grained models that capture the characteristic, malicious behavior of this program. This analysis can be expensive, as it needs to be run only once for a group of similar (or related) malware executable. The key of the proposed approach is that our models can be efficiently matched against the runtime behavior of an unknown program. This allows us to detect malicious code that exhibits behavior that has been previously associated with the activity of a certain malware strain.

Existing System

In the prior work, they proposed a VM commitment system called secom it means secure commitment. It is mainly used to eliminate the malicious state changes in the automatic manner by combining the contents in the OS-level virtual machines to the host. In their proposed work is mentioned in the following options they are clustering the state changes after that have to differentiate the benign and malicious clusters based on that have to commit the benign cluster. For commit the benign cluster first have to found the malicious one that one is takes place by the detection process in the OS in the one by one manner.

Drawbacks

- They did not provide the high performance in the multi virtual machine environment.
- They did not support the malicious activity prevention after detecting the malicious activity.

Proposed System

The multi-secure environment is most needed in the real time scenario. Hence there is a need of the novel frame work for improved detecting and preventing technique in the OS level virtual machines. In this paper they propose the multi secure binder which is called the novel idea for the malicious activity detecting and preventing technique. Here after detecting the particular user they may be able to remove form the process if they perform more malicious activity. Otherwise we compress their file which may act as the supportive file for the other process.

Advantages

- They provide more accurate and less time complexity for the malicious activity detection and prevention process.
- They support the multi OS virtual machine secure commitment.

MODULES

- Establishing Connection for Gather information flow
- Identify Compromised object by grouping clusters
- Develop behavior based malware detection
- Malware detection And Compression

MODULE DESCRIPTION

Establishing Connection for Gather Information Flow

In this module, the client has to register if they did not register before to enter into the process. They have to login then they can write the file with the knowledge of the server, then they can read the file from the server, then they can download the file form the server based on these activities the OS flow is generated automatically. Then the OS flow is created from the server’s system log file.

Identifying Compromised Object by Grouping Clusters

In this module, have to create the cluster based on the each client operations those operations are the read, write, download, directory creation. Based on this here we create four clusters. Then have to label the malicious cluster and the normal cluster for the each cluster. Then have to consider the hash table as the rules and that have to be update into the database.

Develop Behavior Based Malware Detection

In this module, have to commit the malicious cluster based on the hash table which is already updated in the database. Here the malicious cluster detection process takes place by the file extension and also based on the hash table updates those are the client’s behaviors checking process. Such as if they access in the afterhours and from the different access location from the registered location.

Malware Detection and Compression

In this module, they have to collect the malicious cluster detail and the normal cluster detail based on this the normal clusters are recombined again and stored in the server OS flow. Then the malicious clusters have to compress and stored in the database. Then the performance is evaluated based on the time consumption for the process.

CONCLUSIONS

In this paper we propose the novel and enhanced technique to commit and compress the malicious cluster and to make the server more secure than the other system. In the prior work they deletes the malicious cluster files hence there is an inconvenience for the clients and too the server without the file support for the installed software execution. And also here we evaluate the process in the real time scenario by generating the system log file by making the clients to show some activities to create the OS flow those clients are considered as the virtual machines.

REFERENCES

1. W. Sun, Z. Liang, R. Sekar, and V.N. Venkatakrishnan, “One-Way Isolation: An Effective Approach for Realizing Safe Execution Environments,” Proc. 12th ISOC Network and Distributed Systems Symp. (NDSS), pp. 265-278, 2005.
2. S.T. King and P.M. Chen, “Backtracking Intrusions,” Proc. ACMSymp. Operating Systems Principles (SOSP), pp. 223-236, 2003.
3. S. Soltesz, H. Po¨tzl, M.E. Fiuczynski, A. Bavier, and L. Peterson, “Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors,” Proc. Second ACM European Conf. Computer Systems,2007.
4. D. Price and A. Tucker, “Solaris Zones: Operating System Support for Consolidating Commercial Workloads,” Proc. 18th Large Installation System Administration Conf., pp. 241-254, 2004.
5. OpenVZ, “Unique Features of OpenVZ,” http://openvz.org/documentation/tech/features, 2013.
6. SWsoft, “Virtuozzo Server Virtualization,” http://www.swsoft.com/en/products/virtuozzo, 2013.

7. Y. Yu, F. Guo, S. Nanda, L. Lam, and T. Chiueh, “A Feather-Weight Virtual Machine for Windows Applications,” Proc. SecondInt’l Conf. Virtual Execution Environments (VEE), pp. 24-34, 2006.
8. Y. Yu, “OS-level Virtualization and Its Applications,” PhD dissertation, Stony Brook Univ., 2007.
9. Symantec, Inc., http://www.symantec.com/business/security_response/threatexplorer/threats.jsp, 2013.
10. P.-H. Kamp and R.N.M. Watson, “Jails: Confining the Omnipotent Root,” Proc. Second Int’l SANE Conf.,2000.

    Microsoft Security Bull., http://www.microsoft.com/technet/security/current.aspx, 2013.
11. M. Howard, “Fending off Future Attacks by Reducing Attack Surface,” http: //msdn.microsoft.com/en-us/library/ms972812.aspx, 2003.
12. Y. Yu, H.K. Govindarajan, L. Lam, and T. Chiueh, “Applications of Feather-Weight Virtual Machine,” Proc. Int’l Conf. Virtual Execution Environments (VEE),Mar. 2008.
13. R. Paleari, L. Martignoni, E. Passerini, D. Davidson, M.Fredrikson, J. Giffin, and S. Jha, “Automatic Generation of Remediation Procedures for Malware,” Proc. USENIX Conf. Security, Aug. 2010.
14. F. Hsu, H. Chen, T. Ristenpart, J. Li, and Z. Su, “Back to the Future: A Framework for Automatic Malware Removal,”Proc.22nd Ann. Computer Security Applications Conf. (ACSAC),2006.
15. G.W. Dunlap, S.T. King, S. Cinar, M.A. Basrai, and P.M. Chen, “ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay,” Proc. Fifth Symp. Operating Systems Design and Implementation (OSDI),Dec. 2002.
16. Goel, K. Po, K. Farhadi, Z. Li, and E. Lara, “The Taser Intrusion Recovery System,” Proc. 20th ACM Symp. Operating Systems Principles (SOSP),Oct. 2005.
17. N. Zhu and T. Chiueh, “Design, Implementation, and Evaluation of Repairable File Service,” Proc. Int’l Conf. Dependable Systems and Networks (DSN),pp. 217-226, 2003.
18. S.N. Chari and P.-C. Cheng, “Blue Box: A Policy-Driven, Host-Based Intrusion Detection System,” Proc. Symp. Network and Distributed System Security (NDSS),Feb. 2002.
19. S.A. Hofmeyr, S. Forrest, and A. Somayaji, “Intrusion Detection Using Sequences of System Calls,” J. Computer Security, vol. 6,no. 3, pp. 151-180, 1998.
20. M. Christodorescu, S. Jha, S.A. Seshia, D. Song, and R.E. Bryant, “Semantics-Aware Malware Detection,” Proc. IEEE Symp. Security and Privacy, pp. 32-46, May 2005.
21. H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, “Panorama: Capturing System-Wide Information Flow for Malware Detection and Analysis,” Proc. 14th ACM Conf. Computer and Comm. Security(CCS), 2007.
22. PC Magazine, “PC Magazine Benchmarks,”
23. http://www.pcmag.com/encyclopedia_term/0,2542,t=WebBenchi=48947, 00.asp, 2013.

24. Y.-M. Wang, R. Roussev, C. Verbowski, A. Johnson, M.-W. Wu, Y. Huang, and S.-Y. Kuo, "Gatekeeper: Monitoring Auto-Start Extensibility Points (ASEPs) for Spyware Management,"Proc.18th USENIX Conf. System Administration, 2004.

25. Z. Shan, X. Wang, and T. Chiueh, "Safe Side Effects Commitment for OS-Level Virtualization," Proc. Eighth ACM Int'l Conf. Autonomic Computing (ICAC), June 2011.